UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/544,512	04/06/2000	Corneliu I. Lupu	MSFT114614	9057

26389 7590 05/05/2004

CHRISTENSEN, O'CONNOR, JOHNSON, KINDNESS, PLLC
1420 FIFTH AVENUE
SUITE 2800
SEATTLE, WA 98101-2347

EXAMINER	
VU, TUAN A	
ART UNIT	**PAPER NUMBER**
2124	

DATE MAILED: 05/05/2004

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary	Application No. 09/544,512	Applicant(s) LUPU ET AL.	
	Examiner Tuan A Vu	Art Unit 2124	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *04 March 2004*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-18* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-18* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application (PTO-152)
6) ☐ Other: _____.

DETAILED ACTION

1. This action is responsive to the Applicant's response filed 3/04/2004.

As indicated in Applicant's response, claims 2, 7-8, 14 have been amended. Claims 1-18 are pending in the office action.

Claim Rejections - 35 USC § 103

2. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

3. Claims 1, 3-4, 7, 9, 13, 15, and 16 are rejected under 35 U.S.C. 103(a) as being unpatentable over Straub, USPN: 5,430,878 (hereinafter Straub), in view of Nowlin, Jr. et al., USPN: 6,484,309 (hereinafter Nowlin)

As per claim 1, Straub discloses a method for patching a computer application program comprising:

determining whether or not the computer application program is compatible with a computer operating system executing the application (e.g. col. 4, line 4 to col. 5, line 27); and

if the computer application program is determined to be incompatible with the computer operating system, starting a code replacement process to address the incompatible matching(e.g. step *110* -Fig. 1; steps *302, 305* – Fig. 3).

But Straub does not explicitly specify starting up a debugger upon determining an incompatibility with the operating system, although teaches invoking special programs to revise

code and examining code properties (Fig. 1, 2) and to insert instructions at specific points to patch the non-compatible sections of code under revision (e.g. col. 5, line 28 to col. 6, line 25; Fig. 3), i.e. code instructions to identify points at which corrective actions can be effected, i.e. similar to a debug program. Solving program incompatibilities using an external program to identify locations of or gather data from code at which possible corrective actions can be effected was a known concept in the art of software debugging at the time of the invention. In a method to correct application code executing under a different operating system using memory scanning and code replacing analogous to the compatibility code patching by Straub, Nowlin discloses, after knowing that the O.S. system is incompatible with the application (col. 2, line 32 to col. 3, line 34), calling upon a combination of surrogate code and a set of external conversion procedures and DLL filter code to perform application type checking at specific application locations to address the compatibility issues with appropriate patching, i.e. invoking external code to collect information (e.g. *header type* - col. 4, lines 34 to col. 5, line 3) by which corrective actions can be effected, i.e. similar to a debugging process (e.g. col. 2, line 62 to col. 3, line 39; Fig. 2-3; col. 4, line 34 to col. 5, line 42). It would have been obvious for one of ordinary skill in the art at the time the invention was made to apply a combination of dynamic linked code for detecting incompatible data in application code as taught by Nowlin, i.e. start a debugging dynamic linked external program, and add this process to Straub's method of examining specific information relevant to O.S. compatibility issues because according to Nowlin, this would provide smooth and transparent transitions of a wide variety of applications into a more compatible O.S. format without having to recompile or rewrite code thereby

providing a more efficient and/or universal way of translating Window based applications for different platforms (Nowlin: col. 1, lines 27-52).

As per claim 3, Straub further discloses

determining if at least one identifying attribute of a plurality of identifying attributes of the computer application program does not match at least one identifying attribute of a plurality of identifying attributes of compatible applications (e.g. *version numbers*—col. 4, lines 32-54;); and

if at least one of the identifying attributes matches, determining if the computer application program is incompatible, otherwise it is compatible (e.g. col. 5, lines 1-27).

But Straub does not specify matching identifying attributes against attributes of incompatible applications. But in view of the disclosed matching against known patterns by Straub, one of ordinary skill in the art would recognize that the techniques by Straub would have achieved the same results as the claimed limitation suggests; and would also be motivated to modify such pattern matching by comparing such attributes against known incompatible application attributes in case the availability of such incompatible applications is handy for use without additional resources spending because it would have been obvious that providing not only the known compliant but also the known non-compliant patterns, i.e. incompatible attributes as claimed, would make the incompatibility checking even more adequate.

As per claim 4, Straub further discloses storing identifying attributes of compatible applications (e.g. *store table 201* – Fig. 2); and retrieving one of such stored identifying attributes for determining one of the attributes of the computer application program matches the stored attributes of the compatible applications (e.g. col. 5, lines 1-27).

But Straub does not specify storing and using attributes of incompatible applications for the matching against attributes of the target application; but this limitation would have been obvious by virtue of the rationale set forth in claim 3 above.

As per claim 7, this is the computer-readable medium version of claim 1 above, hence incorporates the rejection thereof for the same obvious reasons; and further includes a computer-readable medium to embody the debugging method, which Straub does not disclose. Official notice is taken that the use of a computer-readable medium to store a computer product program code was a well-known concept at the time of the invention. Hence, it would have been obvious for one of ordinary skill in the art at the time the invention was made to use a computer-readable medium to store the computer product program code as disclosed by Straub because this would facilitate the distribution/sale of such computer product and the use of such product by a broader population of computer users.

As per claim 9, this is the computer-readable medium version of claim 3, hence incorporates the corresponding rejections set forth in therein for the same reasons.

As per claim 13, this is the system version of claim 1 above, hence incorporates the rejection thereof for the same obvious reasons.

As per claim 15, in reference to claim 13, this is the computer system version of claim 3, hence incorporates the corresponding rejections set forth in therein for the same reasons.

As per claim 16, this is the system version of claim 4, hence incorporates the corresponding rejections set forth in therein for the same reasons.

4. Claims 2, 5-6, 8, 10, 11-12, 14, 17 and 18 are rejected under 35 U.S.C. 103(a) as being unpatentable over over Straub, USPN: 5,430,878, in view of Nowlin, Jr. et al., USPN: 6,484,309, as applied to claims 1, 7, 13, 15 in view of Preisler et al., USPN: 5,675,803 (hereinafter Preisler).

As per claim 2, Straub discloses specific area of image memory at which the revising process stops and applies patches (e.g. col. 3, line 66 to col. 4, line 31; Fig. 2-3 – Note: interrupting a revision process to insert code is equivalent to setting breakpoint for patching); and Nowlin teaches executing the code reconversion in a debugger with code relocation and entry points information without having to recompile (e.g. col. 6, line 40 to col. 7, line 34).

But Straub (with Nowlin's teaching) does not explicitly disclose setting of breakpoints so to monitor the application via the debugger to determine if at least one such breakpoint has been reached; and applying the patching thereupon; and executing the steps of running the application, monitoring, patching until the application has finished. Official notice is taken that a debug program is being implemented with possibility of setting breakpoints at which data are collected or program execution state can be examined for possible corrective action was a known concept in the art at the time of the invention. Similar to such concept, Straub teaches setting predetermined points at which replacement code can be inserted to solve incompatible instructions (e.g. Fig. 3). And Nowlin teaches calling of code to seek specific data in application header in order to set up patches (col. 4, lines 34 to col. 5, line 3). In the same approach for patching predetermined points of code without compiling as suggested by Straub as to further evidence such known concept, Preisler, in a method to patch target application while checking for run-time errors analogous to the dynamic application patching process by Straub/Nowlin, discloses setting and monitoring for patch sites, i.e. breakpoints while running the application

patch sites -- col. 8, lines 45-65;), and applying the patching process upon such breakpoints (*patch site 20→ Load Instruction*, Fig. 3) and re-executing the steps of monitoring and patching until the application has finished (steps 110, 130, 140, 160 - Fig. 2; col. 6, lines 1-64). It would have been obvious for one of ordinary skill in the art at the time the invention was made to arrange Straub's debugging process so that pre-defined breakpoints as suggested by Preisler and taught by well-known practices are inserted in the process of using patching or filtering as taught by Straub (and further enhanced by Nowlin), and monitor for such breakpoints as taught by known concept during the execution of the application because this would further enhance control over the dynamic state of change of the application program and thereby assert more correctness checking using pre-defined points at which closer data recording or execution code patching as mentioned by Straub/Nowlin or could help debug or apply fix to the application in a more controllable and predictable way.

As per claim 5, Straub in combination with Preisler discloses executing a debugger containing a set or list of breakpoints, each breakpoint having a set of instructions for patching the application (see claim 2); the debugger setting a set breakpoints within the application (see claim 2). Further, Nowlin teaches invoking a dynamic linked filter code used in conjunction with other collaboration/translation code (Figs 2-3) for debugging the application under execution in an heterogeneous O.S., such code being a dynamic linked library (e.g. col. 4, lines 34-41).

But Straub (with Nowlin teachings) fails to disclose accessing the list of breakpoints from such debugger DLL to set breakpoints. However, Preisler discloses setting of patch points while running the debugger (*patch sites* -- cols. 61-65) and Straub suggests stopping points for

the patching process to apply replacement code from the scanning process (e.g. Fig. 3), hence in combination they suggest the teaching as to accessing the list of breakpoints from such debugger. In view of the general knowledge that dynamic linked libraries as used by Nowlin can be compiled externally and used selectively in certain platforms while protected against dynamic modification, it would have been obvious for one of ordinary skill in the art at the time the invention was made to create a debugger as a DLL as suggested by Nowlin and apply this form of DLL debugger to the suggested breakpoint patching techniques of Straub/Preisler and thereby access a list of pre-set breakpoints as suggested above by Straub/Preisler in the intent to implement a breakpoint/patching technique in the debugger. One of ordinary skill in the art would be motivated to do so because, as suggested by the general knowledge that DLL is a form of executable that can be loaded in compatible platform, resistive to changes, and occupying small storage places but highly reusable because of their properties to be linked and dynamically indirectly invoked or wrapped at runtime of a given application, hence facilitate the debugging process and breakpoint presetting of Straub/Preisler in certain operating platforms in which compliance for execution is to be respected; and conformity to such O.S. is what Straub invention is all about.

As per claim 6, Straub (with Preisler's teaching) further discloses upon reaching a breakpoint where compatibility check fails, calling code replacement to patch area instructions set (e.g. Fig. 3), and patching the incompatible application based on such instructions. Straub does not explicitly specify calling a handler associated with a breakpoint; but in view of the use of debugger DLL by Nowlin, this limitation would have been obvious for the same rationale as set forth in claim 1 and/or claim 5 above.

As per claim 8, this is the computer-readable medium version of claim 2; hence incorporates the corresponding rejection set forth therein for the same reasons.

As per claim 10, this is the computer-readable medium version of claim 4, hence incorporates the corresponding rejections set forth in therein for the same reasons.

As per claim 11, this is the computer-readable medium version of claim 5, hence incorporates the corresponding rejections set forth in therein for the same reasons.

As per claim 12, this is the computer-readable medium version of claim 6 above; hence incorporates the corresponding rejections set forth in therein for the same reasons.

As per claim 14, this is the system version of claim 2, hence incorporates the corresponding rejections set forth in therein for the same reasons.

As per claim 17, this is the system version of claim 5, hence incorporates the corresponding rejections set forth in therein for the same reasons.

As per claim 18, refer to claim 6 for the same rationale.

Response to Arguments

5. Applicant's arguments filed 3/04/2004 have been fully considered but they are not persuasive. Following are the reasons therefor.

(A) The Applicants have submitted that Straub and Nowlin do not teach or suggest a debugger to execute the application (Appl. Rmrks, pg. 11, bottom; pg. 12, bottom); and that Nowlin barely teaches a linker to deal with memory issues as opposed to invoking a debugger calls (Appl. Rmrks, pg. 14, bottom para). First, it is noted that the rejection has pointed out that Straub approaches with calls invoked to look for specific properties in the code in order to establish incompatibility issues; and this suggests some routines known in the art of debugging.

Second, Nowlin, after knowing that the O.S. system is incompatible with the application, also invokes a collaboration of dynamic linked programs or conversion code to establish points at which incompatible data are appropriate for patching; hence has suggested routines similar to a debugger calls. Since, both Straub and Nowlin suggest invoking code external to the target application to gather points at which patching can be effected, the concept of starting or running a debugger is strongly suggested upon determining that some incompatibility exists. Third, the fact that added code has to be linked with the original code for the patch to be effective has nothing to do with the approach by Nowlin to identify locations of the target code at which to effect corrective actions because modifying code in order to dynamically resume the execution necessarily involves some context switching or re-linking. Nowlin's teaching provides effects of both debugging with location identification and corrective actions with code modification; one being reminiscent of debug calls, the other of patching for which linking is a necessity. Nowlin's linker actions are only to address the dynamic fix-and-continue process as to optimize processing time as originally intended by Straub's approach via fixing code image instead of recompiling the original application. Besides, the claim does not define what 'debugger' really consists of, and as interpreted, 'debugger' only means an external program to gather data or identify points at which corrective actions can be effected so to remove bugs in the target application, most of such techniques have been shown in Nowlin's method. Hence, as combined, Straub and Nowlin do teach how a dynamic linked debugger (i.e. DLL for patching at specific points) can be usefully evoked along with the application upon determination of an incompatibility issue; and this has fulfilled the claim requirements.

(B) The Applicants have submitted that Nowlin do not teach a DLL with a list of breakpoints for patching the application (Appl. Rmrks, pg. 12, bottom; pg. 13, top); and that Preisler does not teach or remotely suggest adding patches based on application's incompatibility with the operating system (Appl. Rmrks, pg. 13, 3rd para). It is noted that Nowlin does teach applying dynamically linked programs to effect examining of the running program to effect patching as pointed out in the rejection; and that the use of breakpoints in any form of debugger was a well-known concept in software testing. Preisler, therefore, is used to demonstrate how a dynamic patching of an executing program as suggested by Straub or Nowlin can be set with breakpoints as to achieve a time-efficient 'Fix-and-Continue' method; and hence, is not intended to address the limitation of resolving an O.S. incompatibility which is only one instance of use within the methodology of breakpoints-based software patching. Hence, as shown in the rejection, the combination of the suggested techniques by Straub, Nowlin as well as known concepts and Preisler's method has shown that the use of breakpoints in a DLL debugger to be an obvious limitation in view of the motivation as set forth in the rejection.

(C) As per claim 2, Applicants have submitted that Straub does not teach setting breakpoints for outside code invocation in patching the application; that Nowlin only teaches a linker instead of a debugger; and that Preisler does not execute patching at patch sites (Appl. Rmrks, pg. 15, bottom; pg. 16, 2nd and 3rd para). First, sections A and B from above have addressed why Straub's way of revising some specific data in the target application can be complemented with Nowlin's approach in using DLLs for examining points at which patching can take place. Second, Nowlin as mentioned above provides most of the features equivalent of a debugging process and the subsequent invocation of a linker is only an necessary step after the patching has

been done. Third, the concept of debugger has not being defined in the claim as to distinguish what Applicants believe as to be a debugging process versus what Examiner perceives or interprets it to be, because it is the functionality and end result of a debugging process that matters, not the nomenclature used to designate it. And this has been discussed in the above section by means of at least Nowlin's teachings. Fourth, Preisler, as shown in the rejection has clearly applied patches at patch sites (see col. 8, lines 45-65). The claim only recites breakpoints and patching thereupon and as interpreted and explained in the rejection, the combination of Straub/Nowlin along with known concepts and Preisler's teachings have been used to set forth the rationale as to fulfill the required features of the claim, all details therefor having been elaborated in the above section B.

(D) As per the arguments that Straub does not teach matching attributes against incompatible applications of a O.S. (Appl. Rmrks, pg. 17, middle), the rejection has shown how Straub has fulfilled the limitation because matching against compatible O.S. would have made it obvious the process of matching against predetermined non-compliant O.S. data or incompatible applications. Besides, Applicants fail to show specifics on how the rejection amounts to teaching away from the prior art or is at odds with the invention in view of such rationale.

(E) As per arguments on lack in Straub's invention of identifying attributes of incompatible applications (Appl. Rmrks, pg. 18, middle), this falls under the ambit of the arguments already addressed in section D.

(F) As per arguments about claims 5, 11, 17 and 6, 12, and 18, all of these have been addressed in the above section A-C.

Therefore, the claims will stand rejected as in the office action.

Conclusion

6. THIS ACTION IS MADE FINAL. Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the mailing date of this final action.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to Tuan A Vu whose telephone number is (703)305-7207. The examiner can normally be reached on 8AM-4:30PM/Mon-Fri.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Kakali Chaki can be reached on (703)305-9662.

Any response to this action should be mailed to:

Commissioner of Patents and Trademarks

Washington, D.C. 20231

or faxed to:

(703) 872-9306 (for formal communications intended for entry)

or: (703) 746-8734 (for informal or draft communications, please consult Examiner before using this number)

Hand-delivered responses should be brought to Crystal Park II, 2121 Crystal Drive, Arlington. VA. , 22202. 4th Floor(Receptionist).

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

VAT

April 28, 2004